1주차

컴퓨터와 네트워크, 그리고 프로그래밍

지은이: 이은재

01

3시간 워크숍

1. 0:00~0:50: 컴퓨터의 역사와 기본 작동 원리
2. 0:50~1:20: 하드웨어와 소프트웨어
3. 1:20~1:30: 휴식
4. 1:30~2:00: 네트워크와 인터넷
5. 2:00~2:30: 프로그래밍 언어와 개발 과정
6. 2:30~2:40: 휴식
7. 2:40~3:00: 웹사이트가 작동하는 방식과 다음 주 실습 안내

이번 주에 배울 것

1. 컴퓨터의 의미와 탄생
2. 하드웨어와 소프트웨어
3. 네트워크와 인터넷
4. 프로그래밍 언어와 개발 과정
5. 웹사이트가 작동하는 방식

사용된 이미지의 유래와 사용 상태는 이미지 출처 기록에서 확인할 수 있습니다.

// 1. 컴퓨터란 무엇일까요?

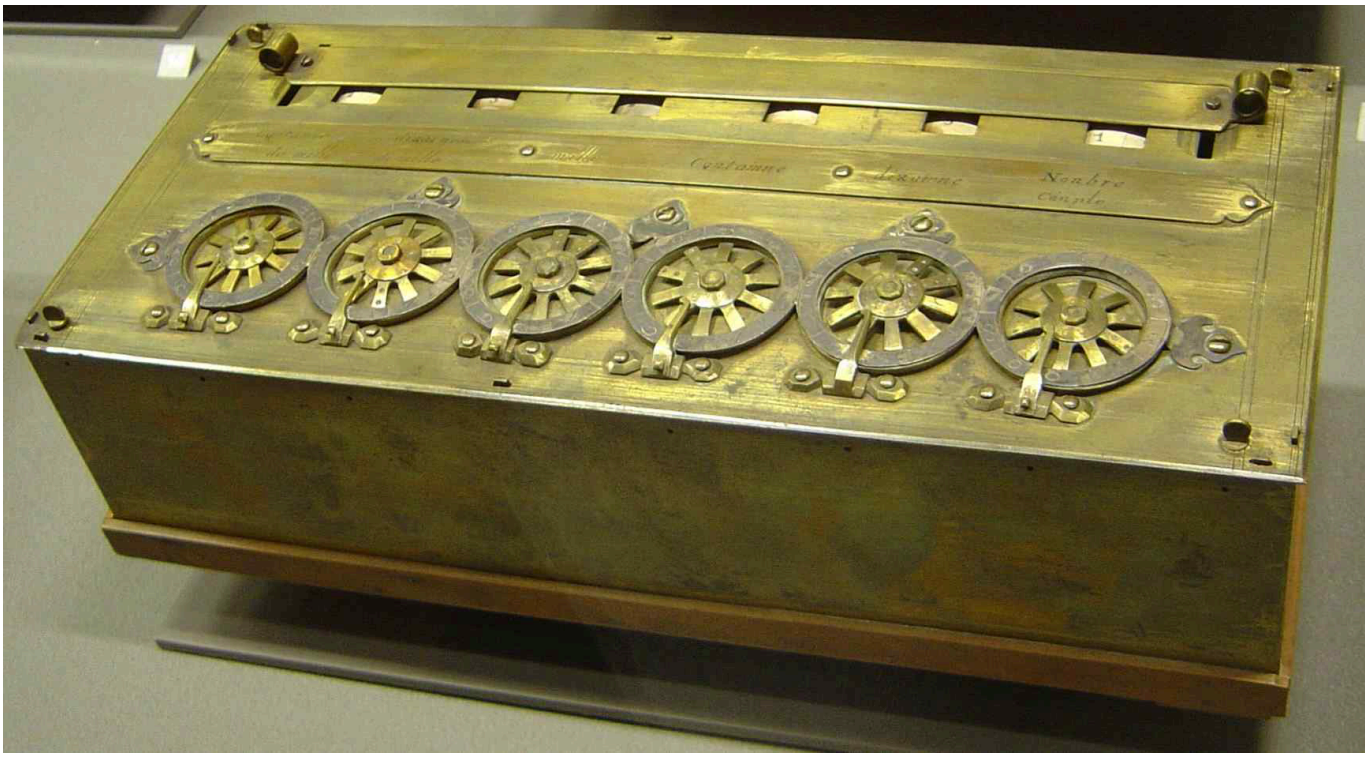
프로그래밍을 본격적으로 공부하고 싶은 사람이라면 그전에 먼저 컴퓨터에 대해서 알아야 할 필요가 있습니다. 프로그램이란 우리가 컴퓨터에 내리는 명령의 집합이고, 프로그래밍도 결국은 그러한 프로그램을 만드는 과정이기 때문입니다.

요리사에게 아무리 귀하고 신선한 식재료가 있어도 처음 보는 것이라 어떻게 손질해야 할지 모른다면 훌륭한 음식을 만들 수는 없겠지요? 마찬가지로 컴퓨터가 무엇이고 어떻게 구성되며, 어떤 일을 할 수 있고 또 어떤 일은 할 수 없는지를 정확히 알지 못한다면 컴퓨터를 제대로 활용하는 일은 불가능할 것입니다.

계산하는 사람에서 계산하는 기계로

우리가 요즘 흔히 말하는 '컴퓨터'는 '개인용 디지털 컴퓨터'를 의미합니다. 말 그대로 이진 논리를 기본 원리로 해서 누구나 사용할 수 있도록 만든 전자계산기입니다. 그런데 컴퓨터 중에는 개인용만 있는 것도 아니고 디지털 컴퓨터만 있는 것도 아닙니다.

전기 없이 톱니바퀴와 나사로만 만든 것, 물로 동작하는 것, 심지어는 실제로 만들어진 적 없이 개념적으로만 존재하는 컴퓨터도 있습니다. 게다가 한때 '컴퓨터'는 전자계산기가 아니라 전문적으로 계산을 해주는 직업을 가진 사람(compute + -er)을 뜻하는 말로 쓰였습니다. 공학용 전자계산기가 보급되기 이전인 1960년대까지만 해도 대량의 복잡한 수학적 연산은 수많은 사람들이 한 방에 모여 기계식 계산기나 주판을 사용해 몇 시간씩 분업해야만 가능했습니다.



파스칼린. 덧셈과 뺄셈을 중심으로 계산했고, 초기 계산기 가운데 비교적 여러 대가 제작되었습니다.

파스칼과 배비지

1623년 빌헬름 슈카르트가 요하네스 케플러에게 보낸 편지와 설계 스케치가 남아 있습니다. 이 기록은 가장 이른 것으로 알려진 사칙연산 기계식 계산기 설계를 보여 줍니다. 그 뒤 블레즈 파스칼이 1642년에 만든 파스칼린은 덧셈과 뺄셈을 중심으로 계산했습니다. 약 50대가 제작되어 소량이나마 여러 대가 만들어진 초기 덧셈 기계라는 점에서 의미가 있습니다. 영국의 찰스 배비지는 1834년, 증기기관에서 나오는 운동에너지를 동력으로 해서 일반적인 수학 계산을 자동화하는 '해석기관'을 구상하기 시작했습니다.

전체 기계는 제작 비용과 당시 공업기술의 한계로 완성되지 못했습니다. 그러나 이 미완의 기계는 처음으로 '프로그래밍 가능한(programmable)' 계산기로 설계되었기 때문에 컴퓨터의 역사에 남았습니다.

배비지가 구상한 해석기관의 일부를 재현한 모형.

생각해 보기

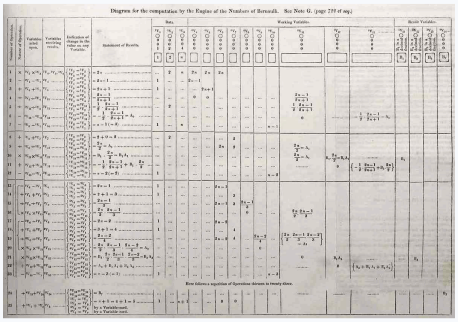
'프로그래밍 가능하다'는 것은 어떤 의미일까요?

에이다 러브레이스

에이다 러브레이스의 초상.

이 만들어지지도 않은 컴퓨터를 바탕으로 초기 프로그래밍 역사에서 중요한 작업을 남긴 사람이 있습니다. 영국 시인 조지 바이런의 딸로도 잘 알려진 에이다 러브레이스입니다. 에이다는 어머니의 뜻에 따라 개인 교습을 포함한 체계적인 수학교육을 받았습니다. 수학자 메리 서머빌의 소개로 1833년 찰스 배비지를 만났고, 그가 보여 준 차분기관의 작동 모형에 깊은 인상을 받았습니다.

그 뒤 에이다는 배비지의 해석기관을 연구했고, 1843년 루이지 메나브레아의 해석기관 논문을 프랑스어에서 영어로 번역하고 원문보다 긴 자신의 주석을 덧붙였습니다. 그 주석에는 해석기관의 베르누이 수 계산 절차가 포함되어 있어 초기 알고리즘 사례로 자주 언급됩니다.

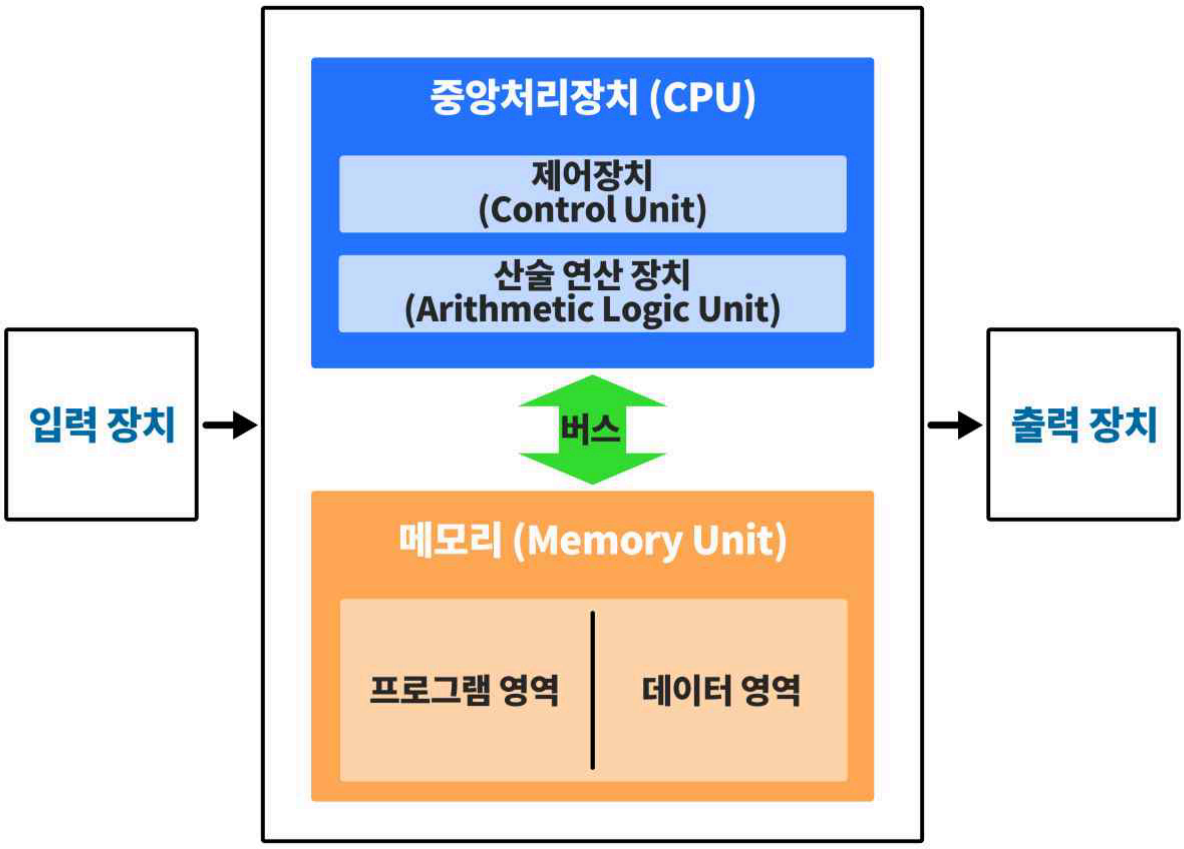


에이다 러브레이스의 주석에 수록된 베르누이 수 계산 절차.

앨런 튜링과 폰 노이만

현대 디지털 컴퓨터의 이론과 구조는 여러 연구자의 작업이 이어지며 발전했습니다. 앨런 튜링은 결정 문제를 연구하며 무엇이 계산 가능한 절차인지, 그리고 그 한계가 어디인지 분석했습니다.

튜링머신은 테이프, 헤드, 상태기록기, 행동표로 이루어져 있으며 알고리즘으로 표현 가능한 계산을 다루는 보편적 계산 모델입니다. 동시에 튜링은 모든 문제를 판정하는 일반 절차가 없다는 사실을 보였습니다.



폰 노이만 구조. 프로그램과 데이터를 메모리에 함께 저장합니다.

프로그램을 메모리에 저장하는 구조 역시 한 사람이 혼자 만든 결과가 아닙니다. ENIAC 뒤의 EDVAC 공동 연구에서 아이디어가 발전했고, 폰 노이만의 「EDVAC 보고서 초안」은 이 구조를 정리해 널리 전파했습니다. ENIAC 같은 일부 초기 전자식 컴퓨터는 새 프로그램을 설정하려면 스위치를 조작하고 케이블을 다시 연결해야 했습니다. 다른 초기 시스템은 천공카드나 종이테이프 같은 외부 매체로 명령을 입력했습니다.

1948년 맨체스터 베이비는 메모리에 저장된 프로그램을 처음 실행한 컴퓨터로 알려져 있습니다. 이후 여러 기계가 이 방식을 발전시켰고, 현대 컴퓨터도 대체로 프로그램과 데이터를 메모리에 두는 저장 프로그램 원리를 이어받았습니다.

// 2. 우리가 사용하는 컴퓨터

이제 실제로 컴퓨터가 어떻게 생겼는지 한 번 알아볼까요? 현대의 디지털 컴퓨터는 크게 하드웨어와 소프트웨어로 구성된다고 할 수 있습니다. 우리의 목표는 소프트웨어를 만들고 제대로 활용하기 위한 내용을 배우는 것이기는 하지만, 소프트웨어도 결국 하드웨어를 제어하기 위한 수단이므로 하드웨어에 관해서도 기본적인 내용을 알아두는 것이 좋습니다.

하드웨어

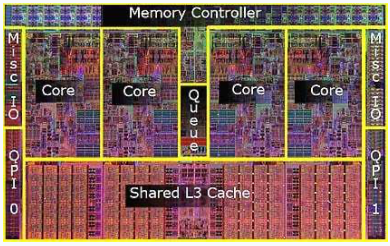
CPU (Central Processing Unit)

CPU의 실제 계산 회로는 실리콘으로 만든 작은 다이(die) 하나 이상에 수십억 개의 트랜지스터를 새겨 구성합니다. 트랜지스터로 만든 연산 회로가 모여 코어를 이루고, 한 다이에는 코어가 하나 이상 들어갈 수 있습니다. 다이는 외부 연결부와 보호 구조를 갖춘 패키지 안에 들어가며, 우리가 메인보드에 장착하는 CPU는 이 패키지 전체입니다. 프로그램을 해석하고 데이터를 처리하고 다른 부품들을 제어하는 데에 필요한 모든 연산을 담당하는 부분이기 때문에 컴퓨터의 성능을 가장 크게 좌우하기도 합니다.

클럭(clock)은 CPU가 1초에 반복하는 동작 주기의 수를 말하며 GHz로 표시합니다. 하지만 실제 성능은 클럭과 코어 수만으로 정해지지 않습니다. 한 주기에 처리할 수 있는 명령어 수(IPC), 프로세서 구조, 캐시의 크기와 효율, 실행할 작업 특성도 함께 영향을 줍니다. 예전에는 CPU가 하나면 코어도 하나였지만, 하나의 코어만으로는 전력 소비나 발열 같은 물리적 문제 때문에 성능 향상에 한계가 있어 현재 대부분의 CPU는 여러 코어가 동시에 작업을 처리하는 방식을 사용합니다.

CPU 다이를 보호하고 메인보드와 연결하는 인텔 CPU 패키지.

연산 회로가 든 다이를 내부에 포함한 AMD CPU 패키지.



CPU 다이 안의 코어와 캐시 영역을 나타낸 도면.

RAM (Random Access Memory)

CPU가 바로 사용할 프로그램과 데이터를 잠시 올려두는 작업 공간입니다.

데스크톱 컴퓨터에 장착하는 RAM 모듈.

기판에 직접 장착된 메모리 칩도 CPU의 임시 작업 공간으로 쓰입니다.

HDD / SSD (Hard Disk Drive / Solid State Drive)

전원을 끈 후에도 프로그램과 데이터를 유지하는 저장장치입니다.

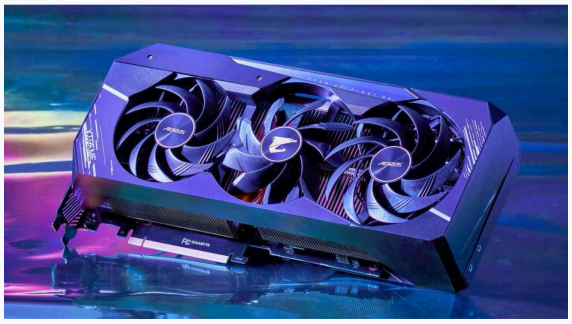
금속 케이스로 내부를 보호한 HDD의 외부.

회전하는 자기 디스크에 데
이터를 기록하는 HDD.

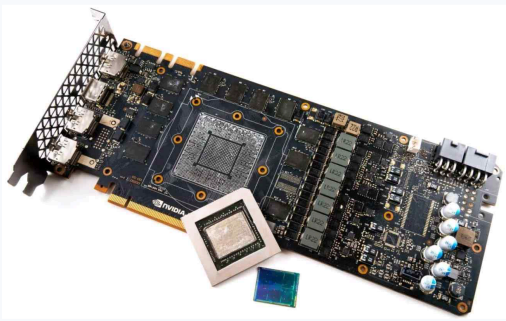
반도체 메모리에 데이터를 기록하는 SSD.

GPU (Graphics Processing Unit)

화면을 그리는 데 필요한 많은 계산을 동시에 처리하는 장치입니다.



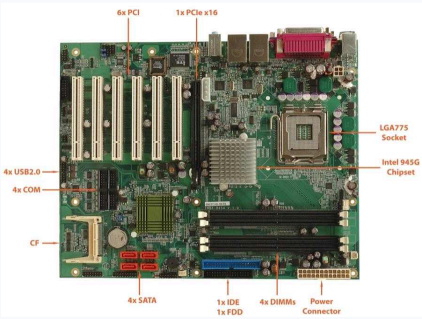
GPU와 냉각 장치 등이 결합된 그래픽 카드.



냉각 장치 안쪽의 GPU와 메모리 칩, 기판.

Main Board

CPU, RAM, 저장장치, GPU와 입출력 장치가 서로 데이터를 주고받도록 연결합니다.



메인보드는 CPU, RAM, 저장장치, GPU, 전원을 서로 연결합니다.

Power Supply / Battery

각 부품이 필요로 하는 형태로 전기를 공급합니다.

데스크톱 컴퓨터의 전원공급장치.

노트북 내부에 장착되는 배터리.

소프트웨어

OS (Operating System)

여러 데스크톱과 모바일 운영체제를 상징하는 로고.

운영체제라고도 부르는 OS는 하드웨어와 소프트웨어 모두를 효율적으로 관리할 수 있도록 도와주는 소프트웨어입니다. 전원 버튼을 누르면 컴퓨터가 켜지는 것부터 시작해서 모든 작업을 마치고 시스템을 종료하는 것까지 어느 하나도 OS 없이는 동작하지 않습니다.

커널(kernel)은 CPU, 메모리, 저장장치, 입출력 장치 같은 핵심 자원과 하드웨어를 관리하는 OS의 중심 부분입니다. 셸(shell)은 사용자의 명령을 해석해 OS 기능을 요청하거나 프로그램을 시작하도록 돕는 인터페이스입니다. 응용프로그램 실행 전체를 셸이 담당하는 것은 아닙니다. 프로그램은 OS가 제공하는 기능을 이용해 동작합니다. 가장 대중적으로 사용되는 OS는 Windows와 macOS, 그리고 Linux이지만 하드웨어의 특성과 용도에 따른 수많은 OS가 사용되고 있습니다.

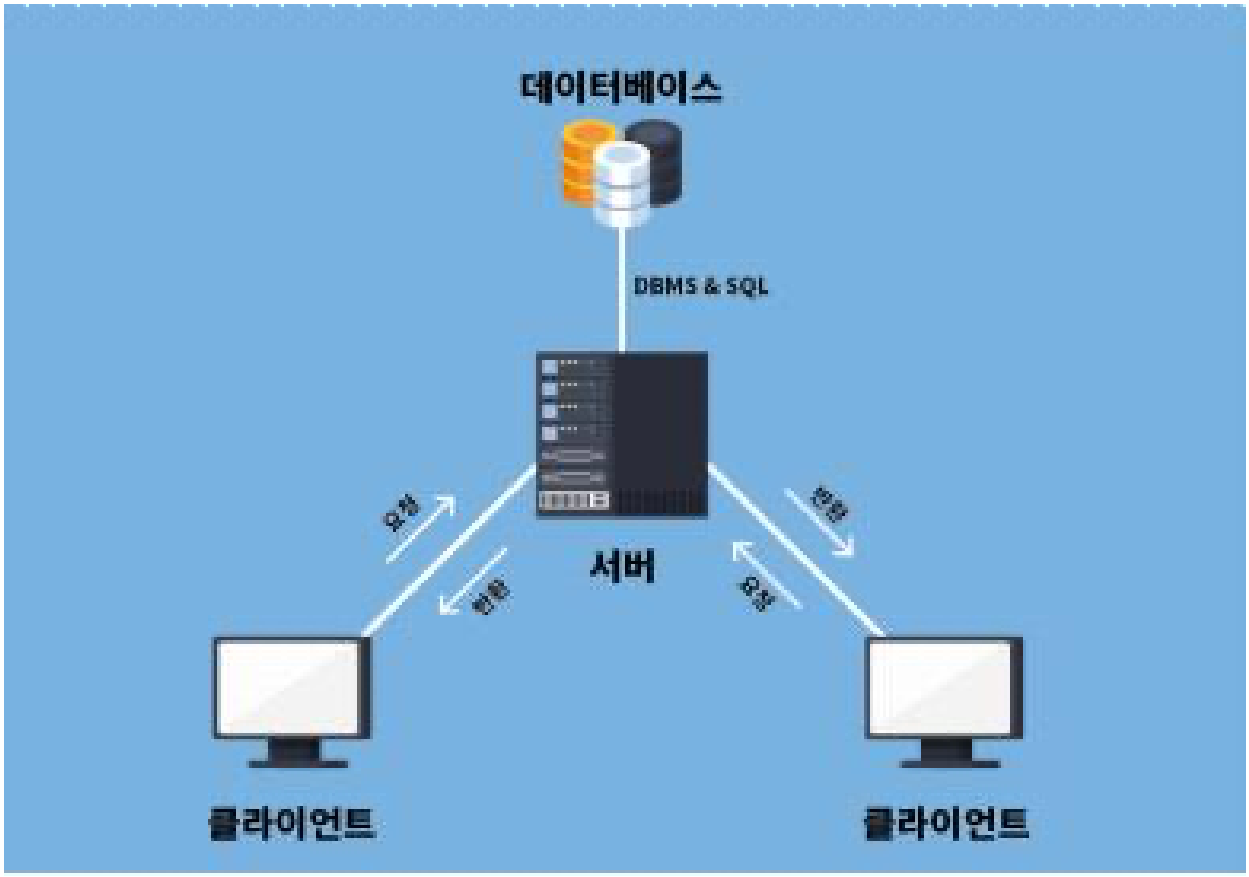
응용프로그램 (application software)

응용프로그램은 일반적으로 컴퓨터에서 실행되는 소프트웨어 중에서 OS를 제외한 나머지를 가리킵니다. 이름이 '응용'인 이유는 시스템 소프트웨어인 OS의 기능을 이용해서 사용자가 원하는 특정한 기능만을 수행하도록 만들었기 때문입니다. 컴퓨터 입장에서 보자면 응용프로그램은 OS의 보조적 기능이라고 할 수 있습니다.

하지만 사용자 입장에서는 컴퓨터를 사용하는 주목적이 바로 응용프로그램을 사용하기 위한 것이기 때문에 '프로그램'이라고 하면 보통 응용프로그램을 의미하게 되었습니다.

// 3. 네트워크

네트워크란?



클라이언트가 요청하면 서버는 데이터를 처리해 응답합니다.

네트워크는 여러 군데에 분산되어 있는 컴퓨터들이 서로 데이터를 공유할 수 있도록 하나의 통신망으로 연결한 것을 말합니다. 일상에서 가장 많이 사용하는 네트워크는 물론 인터넷입니다. 하지만 네트워크에는 인터넷만 있는 것은 아닙니다.

사용하려는 목적과 범위, 용도에 따라서 네트워크 구축에 사용하는 통신 방법과 그 규모가 달라집니다. 각각의 컴퓨터를 직접 케이블로 연결하거나 간단한 라우터만을 사용하는 소규모 네트워크도 있지만, 서버용 컴퓨터를 사용하거나 전용 데이터 센터를 새로 짓기도 하는 방대한 네트워크도 있습니다.

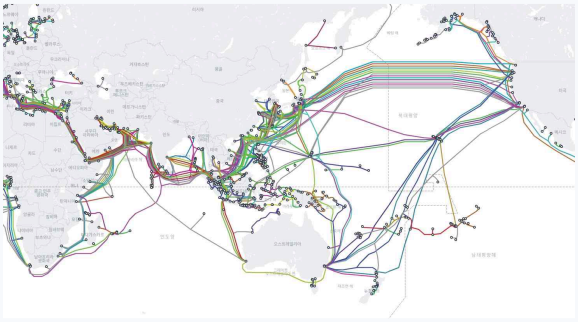
클라이언트와 서버

우리가 인터넷을 사용할 때 컴퓨터와 네트워크에서는 어떤 일이 일어나고 있는 것일까요? 지금부터 우리가 어떤 홈페이지에 접속해서 정보를 조회한다고 해봅시다. 이때 내가 사용하는 컴퓨터를 '클라이언트'라고 부르고 접속하려는 홈페이지의 정보를 담고 있는 컴퓨터를 '서버'라고 부릅니다.

각각의 컴퓨터에는 고유한 주소가 있습니다. 이때 서로 약속한 통신 형식인 '프로토콜'에 따라서 접속하고자 하는 서버의 주소로 원하는 정보를 전송해달라는 요청을 보냅니다. 서버는 받은 요청에 문제가 없는지 확인하고 데이터베이스의 정보를 가공한 뒤 그 결과를 클라이언트에게 전송합니다. 이 요청과 응답을 사람이 쉽게 이용할 수 있도록 화면으로 보여주는 소프트웨어가 우리가 매일 사용하는 웹 브라우저입니다.

인터넷

PAN, LAN, MAN, WAN은 네트워크가 닿는 규모와 범위를 설명하는 분류입니다. 인터넷은 규모가 다른 수많은 네트워크가 공통 프로토콜로 서로 연결된 전 지구적 네트워크입니다. ARPA넷은 미국 고등연구계획국의 지원으로 연구기관 간 컴퓨터 자원을 공유하기 위해 구축된 패킷 교환 네트워크였습니다. 1969년에 운영을 시작한 이 네트워크는 현대 인터넷의 전신 중 하나로 평가됩니다.



인터넷 통신의 많은 부분은 대륙과 대륙을 잇는 해저 케이블을 통해 이동합니다.

바다 밑을 지나는 해저 통신 케이블.

광섬유를 물리적 손상에서 보호하는 해저 케이블의 구조.

웹(WWW)은 인터넷과 같은 규모 분류가 아니라, 인터넷 위에서 동작하는 정보 서비스입니다. URL로 문서의 위치를 가리키고 HTTP로 문서를 주고받으며, 브라우저가 문서를 읽고 링크를 따라 이동하게 합니다.

팀 버너스리와 웹의 탄생

영국의 컴퓨터 과학자 **팀 버너스리(Tim Berners-Lee)**는 인터넷을 발명한 사람이 아니라, 이미 존재하던 인터넷 위에서 정보를 연결하고 읽는 **웹(WWW)**을 설계한 인물입니다.

보르헤스가 먼저 상상한 갈라지는 읽기

아르헨티나 작가 **호르헤 루이스 보르헤스**의 1941년 단편 「갈림길이 있는 정원」(The Garden of Forking Paths)은 하나의 이야기가 여러 갈래의 경로와 가능성을 품는 미로를 그립니다. 그래서 훗날 이 작품은 독자가 링크를 따라 다른 경로로 이동하는 **하이퍼텍스트의 문학적 선행 이미지**로 자주 읽혔습니다.

다만 팀 버너스리가 이 작품에서 **직접 영감을 받아** 웹을 제안했다고 확인된 기록은 찾기 어렵습니다. 확인되는 직접 배경은 CERN 여러 연구기관에 흩어진 정보를 연결하고 공유하려는 문제였습니다. 따라서 보르헤스의 작품은 웹의 직접 발명 계기라기보다, 비선형적인 연결 구조를 먼저 상상한 문학 작품으로 구분해 이해하는 편이 정확합니다. MIT Press의 『The New Media Reader』 소개 보기

1. **1989년:** CERN에서 여러 연구기관의 정보를 자동으로 공유하기 위한 인터넷 기반 하이퍼텍스트 시스템을 처음 제안했습니다.
2. **1990년 말:** HTML, HTTP, URL이라는 핵심 요소와 최초의 웹 서버, 브라우저 겸 편집기를 구현했습니다. 최초의 웹사이트 주소는 `info.cern.ch` 였습니다.
3. **1993년:** CERN이 웹 소프트웨어를 로열티 없이 공개하면서 누구나 사용하고 개선할 수 있는 기반이 마련되었습니다.

CERN의 웹 탄생 기록 자세히 보기

확인해 보기
[최초의 웹사이트(info.cern.ch) 바로 열기]

1. **macOS**: Apple 메뉴 → 시스템 설정 → 네트워크 → 현재 연결된 Wi-Fi 또는 Ethernet → 세부사항 → TCP/IP에서 **IPv4 주소**를 확인합니다.
2. **Windows 10·11**: 시작 → 설정 → 네트워크 및 인터넷 → Wi-Fi → 현재 연결된 네트워크의 속성에서 **IPv4 주소**를 확인합니다. 유선 연결이라면 Wi-Fi 대신 Ethernet을 선택합니다.

확인할 값: 이 실습에서는 같은 공유기나 사내 네트워크 안에서 사용하는 **로컬 IPv4 주소**를 찾습니다. 보통 `192.168.x.x` 또는 `10.x.x.x` 처럼 보이며, 인터넷에 표시되는 공인 IP 주소와 다를 수 있습니다.

// 4. 프로그래밍과 언어

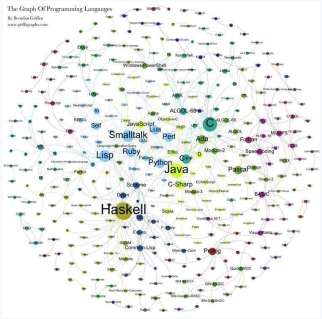
프로그래밍 언어

사람이 한국어나 영어 같은 자연어를 사용하는 것처럼 컴퓨터도 컴퓨터가 알아들을 수 있는 언어인 기계어를 사용합니다. 프로그래밍 언어는 추상화된 정도에 따라서 그 계층이 달라집니다. 계층이 가장 낮은 언어, 즉 실제로 기계가 정보를 기록하는 데에 사용하는 언어는 0과 1로 이루어진 '바이너리 코드'입니다.

'컴퓨터는 0과 1밖에는 모른다'는 이야기가 알려진 것도 이 때문입니다. 우리가 컴퓨터에게 어떤 프로그램을 실행하라고 지시하려면 컴퓨터가 이해할 수 있는 기계어로 입력해 주어야 할 것입니다.

문제는 우리가 바이너리 코드를 이해하지 못한다는 데에 있습니다. 초창기의 프로그래머들은 정말로 바이너리 코드로 프로그램을 만들었지만, 다행히도 현대에는 더 이상 바이너리 코드를 사용하지 않고도 프로그램을 만들 수 있도록 도와주는 다양한 고급 언어들이 개발되어 있습니다.

언어의 계층과 역할
기계어와 어셈블리어처럼 하드웨어에 가까운 로우레벨 언어가 있고, C, Java, JavaScript, Scratch처럼 사람이 다루기 편하도록 추상화한 하이레벨 언어가 있습니다. 언어는 각각 표현 방식과 잘하는 일이 다르므로, 해결할 문제에 맞게 선택합니다.



프로그래밍 언어는 서로 영향을 주고받으며 다양한 목적으로 발전했습니다.

같은 결과, 다른 언어

화면에 같은 문장을 출력하더라도 언어마다 문법과 실행 방식이 다릅니다. 아래 예시에서 공통점과 차이점을 찾아봅시다.

코드 0-1 'Hello World!'를 출력하는 C 코드

```c
#include <stdio.h>
int main(int argc, char *argv[])
{
    printf("Hello World!\n");
    return 0;
}
```

↳ 실행화면

```
Hello World!
```

C 언어의 Hello World 예시.

코드 0-2 'Hello World!'를 출력하는 자바 코드

```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

↳ 실행화면

```
Hello World!
```

Java의 Hello World 예시.

코드 0-3 'Hello World!'를 출력하는 파이썬 코드

```python
print('Hello World!')
```

↳ 실행화면

```
Hello World!
```

스크립트 언어의 Hello World 예시.

프로그래밍 과정

프로그램은 코드를 쓰는 일만으로 완성되지 않습니다. 문제를 명확히 정하고, 해결 절차를 설계하고, 코드로 옮긴 뒤 반복해 검증하고 관리해야 합니다.

문제 인식

해결해야 할 문제가 명확하지 않으면 일단 코드를 작성할 이유가 불분명해지기 때문에 코딩 자체가 길고 지루한 작업이 됩니다. 그 결과로 복잡하고 비효율적인 문장들을 사용하게 되는데, 흥미 면에서도 성능 면에서도 결코 좋은 일이 아닙니다. 자신이 어떤 문제를 해결하고 싶은지 확실히 정해두도록 합시다.

의사코드(pseudo code) 작성

프로그래밍 언어로 코딩을 하기 전에 미리 프로그램을 설계해 보는 단계입니다. 이때는 프로그래밍 언어를 사용할 필요 없이 자연어나 기타 각자에게 편한 방식을 고르면 됩니다. 굳이 할 필요가 있을까 싶지만 의사코드를 작성해 보느냐에 따라 실제 코딩 시간이 길어지기도, 줄어들기도 합니다.

```
사용자가 버튼을 누른다
만약 설명이 닫혀 있다면
  설명을 연다
그렇지 않다면
  설명을 닫는다
```

코딩

실제 프로그램을 작성하는 과정입니다. 영화에서 보던 멋진 해커의 화려한 모습은 사실과는 거리가 아주 멉니다. 사실은 멍하게 모니터를 보고 생각하는 시간이 더 많습니다.

컴파일 (compile)

우리가 작성한 코드를 컴퓨터가 실행하도록 처리하는 방식은 언어와 실행 환경에 따라 다릅니다. 어떤 언어는 컴파일러가 코드를 미리 기계어나 중간 형태로 번역하고, 어떤 언어는 인터프리터가 실행하면서 해석합니다. 브라우저의 JavaScript 실행 환경처럼 해석과 실행 중 컴파일을 함께 사용하는 경우도 있습니다.

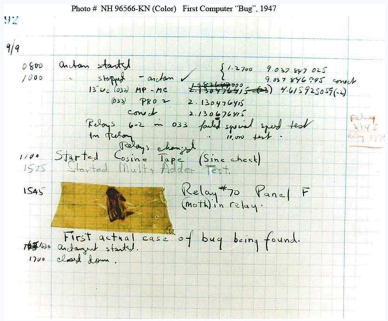
빌드 (build)

빌드는 소스 코드와 자원을 실행하거나 배포할 수 있는 형태로 준비하는 과정입니다. 언어와 실행 환경에 따라 빌드 결과는 달라지며, 항상 하나의 실행 파일을 만드는 것은 아닙니다. HTML, CSS, JavaScript로 만든 간단한 웹사이트는 세 파일을 그대로 브라우저에서 열어 실행할 수도 있습니다.

디버깅

1차로 완성된 프로그램이 오류 때문에 아예 실행되지 않거나 실행은 되더라도 원하는 결과를 출력하지 못할 경우, 프로그램에 '버그'가 있다고 표현합니다. 그리고 버그를 찾아 고치는 것을 '디버깅'이라고 부릅니다.

짧은 코드일 때는 오류의 원인을 금방 찾을 수 있지만 코드가 길어질수록 오류를 찾기가 대단히 어려워집니다. 그래서 디버거라는 프로그램의 도움을 받는데, 디버거는 우리가 작성한 프로그램을 한 줄씩 실행하여 문제가 있는 부분을 표시해 줍니다. 그렇게 표시된 버그는 다시 적절한 코드로 수정하고 새로 빌드하면 문제가 대부분 해결됩니다.



1947년 Harvard Mark II 작업 일지에 붙인 나방과 기록. 이 사례는 컴퓨터 버그에 관한 유명한 기록으로 남았습니다.

테스트

테스트는 프로그램이 기대대로 동작하는지 확인하고 결함 가능성을 줄이는 과정입니다. 하지만 준비한 테스트를 모두 통과했다고 해서 버그가 전혀 없음을 증명하지는 못합니다. 테스트가 통과되어도 성능이 더 좋다거나 디자인적인 측면에서 훌륭하다는 것이 저절로 보장되는 것도 아닙니다.

따라서 출시를 목적으로 하는 대부분의 프로그램은 다양한 사용자들에게 미리 테스트를 받는 과정을 거칩니다. 그 결과를 바탕으로 제작자와 사용자 모두가 만족할 때까지 수정이 이루어집니다. 프로그램이 어느 정도 완성되었는지, 그리고 얼마나 개방적인 테스트인가에 따라 프리알파, 알파, 베타 테스트 등으로 나뉘어 진행됩니다.

배포(릴리즈)

프로그램을 사용자가 실제로 쓸 수 있도록 세상에 내놓는 단계입니다. 배포된 프로그램은 수많은 사람들의 컴퓨터에 설치되어 사용될 수 있기 때문에 여러 버전의 배포 후보를 개발해 두고 어떤 것으로 배포할지 신중히 결정합니다.

버전 관리

프로그램이 한 번 출시되었다고 해서 개발자의 역할이 끝나는 것은 아닙니다. 테스트에서는 발견하지 못한 문제들이 생길 수도 있고 프로그램에 기능을 추가하거나 빼야 할 수도 있습니다. 업데이트할 때마다 어떤 부분이 어떻게 바뀌었는지 정확히 기록해 두지 않으면 점점 관리하기가 어려워집니다.

유지보수

프로그램이 배포된 후로 시간이 지나면 호환성이나 성능, 보안상의 다양한 문제가 발생할 수 있습니다. 따라서 지속적인 유지보수가 필요한데, 내가 개발한 것을 내가 고치는 것조차도 굉장히 힘든 작업입니다. 하물며 다른 사람이 작성한 코드를 아무런 설명도 없이 스스로 이해해서 고치는 것은 거의 불가능에 가깝습니다.

개인 프로젝트라면 상관이 없을 수 있지만 협업을 해야 하는 상황이라면 주석과 별도의 문서로 프로그램 개발과 관련된 사항들을 정리해 두어야 합니다.

// 5. 웹사이트는 어떻게 작동할까요?

앞에서 살펴본 클라이언트와 서버, 프로그래밍 언어의 역할을 웹사이트에 연결해 봅시다. 브라우저는 서버에서 받은 파일을 해석해 화면을 만든다.

- HTML은 콘텐츠의 구조를 표현한다.
- CSS는 화면의 모양과 배치를 정한다.
- JavaScript는 사용자 행동에 반응하는 동작을 만든다.

다음 주부터 세 파일을 직접 만들고 하나의 웹사이트로 연결한다.